HOLMES: An event-driven solution to monitor data centers through continuous queries and machine learning

Keywords
Complex Event Processing, Messaging-oriented Middleware, Data Streams, Anomaly Detection, Monitoring

ABSTRACT
Supervisory processes are fundamental when running data center operations striving for fault resilience: any downtime can directly affect the business’s income and definitely its reputation. Current monitoring tools rely on experts to configure constant thresholds on single streams, which is not appropriated for dynamic systems and insufficient to capture complex patterns. We present HOLMES, built to support data center experts to anticipate failures with a solution that combines Event Driven Architecture, Complex Event Processing and an unsupervised machine learning algorithm. Based on rules created by the users, the system continuously checks for known problems. Meanwhile, for the unknown ones, we leverage the CEP engine for aggregating and joining streams of real-time data to feed normalized input to FRAHST, our machine learning algorithm that detects anomalous patterns across multivariate numerical streams. We describe how the UI module also operates within the publish/subscribe paradigm to enhance situational awareness. The system had very well acceptance and was successfully implemented at one of the largest Internet Service Providers in South America.

1. INTRODUCTION
In this paper, we present HOLMES, our system that combines machine learning algorithms with Event Stream Processing to leverage the processing of the incoming raw data. In the first session, an overview of the system architecture is presented with all its relevant modules. Next, we describe our Complex Event Processing usage and our machine learning integration. Some aspects of the visualization module and the User Interface follows and finally, a successful user case implementation is briefly related.

1.1 Data centers
Our target domain is an Internet Service Provider’s (ISP) data center, where huge volume of real-time data needs to be monitored by operations team with the objective to maintain high-availability and quality of the services.

A data center is a facility that houses computer clusters and their associated components, such as storage and telecommunications subsystems. ISPs usually offer a wide range of online services such as e-mail, games and news portal. In order to serve a large number of concurrent users, requests to a web application are balanced across many servers to ensure low latency, hence a good user experience. The heterogeneous nature a typical web application, illustrated in Figure 1, reflects on a complex IT infrastructure relying on the integration between various software, middleware and hardware components.

![Figure 1: Typical web application architecture.](image)

Every server, application and network device exports statistics that can be collected and monitored. Metrics are usually collected every minute so as to reduce load on the network,
but some operations may require higher resolution.

1.2 Motivation

The complexity of data centers poses many challenges for system administrators, who must constantly monitor their infrastructure to provide appropriate quality of service to their users.

Our goal is to devise a system that can detect anomalies as soon as possible in the infrastructure in order to minimize losses. Alarms should increase the situational awareness of human experts who can check the system and promptly act when needed and are spared from the rather implausible burden of continuous monitoring.

Operators have long relied upon monitoring software to analyze the current state of the infrastructure, nevertheless it is still very difficult to effectively monitor an entire data center. Traditional monitoring software, such as Cricket and Nagios, requires the manual configuration of thresholds for every data stream to be monitored: an expert spends significant amount of time to define what is considered normal behavior for each data stream using a constant threshold value. This strategy is very poor and does not take advantage of all data available, specially because several measurements from distinct sources are strongly correlated – for example, redundant network links sharing traffic and different machines running same applications with load balancing. Furthermore, many alerts associated with individual streams may be symptomatic of the same failure and with the advent of hardware virtualization, the manual configuration of these threshold becomes even more impractical and the need for better tools is evident.

This work is motivated by the real need to improve the current monitoring solution at the data center of one of the largest Internet Service Provider (ISP) in South America – it has over 1,200 servers generating roughly 40,000 charts which operators need to browse in order to explore the current state of the infrastructure. A more intelligent monitoring system should decrease the time to detect faults, less the burden on the operations team and provide additional contextual information when an anomaly occurs to make troubleshooting more efficient.

2. SYSTEM ARCHITECTURE

The main goal of the architecture is to allow a high rate of events to be processed and analyzed through a combination of decoupled strategies, based on CEP and machine learning algorithms.

We devise a loose-coupled event-driven architecture where all main modules are publishers or subscribers to an event broker, also known as event/message bus. The main structure of the system is illustrated in Figure 2.

The system has four modules: CEP engine, machine learning, visualization and storage. A CEP engine is responsible for the correlation of events through user-defined queries that allow known problems to be captured. The CEP engine handles internal system events (such as generated alerts) and normalizes events for the machine learning module. The web server listens to specific queues and broadcasts visualization events to the connected web clients to feed real-time charts.

The storage module is responsible for persisting model data as well historical data, but more detailed discussion is beyond the scope of this paper.

2.1 Message Oriented Middleware

The message broker plays an important role in the system architecture, because it centralizes all communication between systems modules. In order to prepare for maximum scalability and modularity, a message oriented middleware (MOM) enables event-driven applications to communicate based on the publish/subscribe. We chose ActiveMQ since JMS is the standard for MOM and provides many possible transport and message protocols for integration. Besides JMS, the non-JVM components rely on the Streaming Text Orientated Messaging Protocol (STOMP) for efficient communication.

2.2 Data source integration

The most important components in a data center can be monitored using the Simple Network Management Protocol (SNMP), which include routers, access servers, switches, bridges, hubs, temperature sensors and computer hosts. We integrate with Nagios in order to reuse the existing data collection plugins to enable a wide variety of data to be available into our system. We hook a custom plugin into Nagios Event Broker, and send normalized JavaScript Object Notation (JSON) events to our MOM infrastructure.

3. COMPLEX EVENT PROCESSING

Event processing is used since 1960’s, in the context switching of threads and processes. In 1970’s, TCP/IP protocol brought a new significant idea of event abstraction and the subsequent ubiquitous explosion of networking, sensor devices, and the Internet caused an increasing demand for...
real-time processing. Some kind of more complex event processing was definitely established.

Nowadays, with its maturing, Complex Event Processing technology has become the choice for many monitoring applications. Beginning in 2000 with financial trading and extending to many industrial sectors and applications such as Business Process Management, sensor network applications – RFID reading, web mining, fraud detection, data center monitoring, logistic tracking and many others.

A lot of information passes through the intricacies of the network and CEP offers a set of concepts to deal with this apparently chaotic scenario. Indeed, extracting relevant information from an event cloud is of fundamental importance in striving for faults and abnormal conditions in a data center. With CEP we can aggregate and associate real-world events to abstractions, e.g., a business rule, providing a more human modeling and a more efficient feedback to the data center expert.

We chose Esper as our CEP engine, since it is an excellent opensource project, offering an expressive temporal query language and can be easily embedded into an application.

3.1 Esper

In the 1980’s, active databases technology were implemented to satisfy the real-time processing need, stimulating the creation and developing of languages to express rules and event patterns. Today, we can find many computer languages dedicated to event processing well-founded in some kind of a SQL-like language, such as Event Processing Language from Esper project.

Esper is an Event Stream Processing engine written in Java, an implementation of CEP concepts and ideas. Its main task is to provide an event stream processing module, including event representations and event pattern matching, supporting the extraction of meaningful information from the heavy amount of data within a stream. However, in spite of deal with a classical concept of database, we take advantage of non-persistent data. While the events pass through memory, the query engine continuous sieves for the relevant ones, saving processing time. This technique is also known as continuous query. Then, accomplishing event pattern matching requires defining a query in EPL, e.g.:

```
SELECT slope, usage
FROM CPU.win.time(10 sec).stat:linest(timestamp, user),
Disk.win.time(10 sec) WHERE slope >= 0
AND usage > 100
```

Figure 3: Example of EPL query to detect when the CPU usage is increasing.

This query attempts to detect variations in a CPU within a 10 seconds time window and defines a specific threshold for Disk usage monitoring. In our system, similar queries can be written by users and stored as rules. These rules are processed and might generate derived events, such as alerts. Alerts then should be used for troubleshooting, increasing reactive actions in dealing with some well-known problems.

4. MACHINE LEARNING

While hand-crafted continuous queries are used to automate known troubleshooting documents, we rely on machine learning to detect never seen before anomalies.

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior according to both local and temporal contexts.

In data centers, an anomaly is a short-lived deviation from its normal operation. Common source of anomalies are: software bugs (e.g. memory leaks, not optimized database queries), hardware malfunctioning (e.g. disk failures) or faults in the underlying subsystems (e.g. broken communication network access link).

A recent survey [1] note that anomaly detection has traditionally dealt with record or transaction type data. They further indicate that most techniques require the entire test data before detecting anomalies, and it mentions very few online techniques. Indeed, most current algorithms assume the dataset fits in main memory [12]. Both aspects violate the requirement for real-time monitoring data streams.

The high dimensionality of the observation data, together with the frequent changes of system normal conditions resulting from user behavior as well as from changes in the infrastructure itself makes detection even more difficult.

4.1 FRAHST

We use FRAHST [11] (Fast Rank-Adaptive row Householder Subspace Tracking) as our unsupervised streaming algorithm for anomaly detection. In this technique, just one pass over the data is allowed. It tracks the principal subspace of r dimensions from N numerical data streams. It automatically learns the principal subspace from N numerical event streams and an anomaly is indicated by a change in the number of latent variables.

FRAHST provides state-of-the-art estimates for the subspace basis, and successfully detects subtle anomalous patterns. For more information, see [10]. The anomaly detection procedure follows directly from the rank-adaptive nature of FRAHST. Alarms are raised whenever there is an increase in rank, as laid out below.

```
last ← 0
for t = 1, 2, . . .
    FRAHST(t)
    if r(t) > r(t − 1)
        if t > last + 1: raise an alarm at t
        last ← t
```

Figure 4: Anomaly detection routine

The control variable last is used to suppress alarms when there are rank increments in consecutive interval, which are likely to be false alarms. The intuition is that a change in the underlying streams might be so great, that more than one additional latent variables are necessary to achieve the same reconstruction error.

4.2 Integration

More specifically, the data center operator chooses a set of streams to be monitored together and a query can be placed to join the corresponding underlying streams in order to produce the input vector z(t) for our algorithm at periodic equidistant intervals – the join operator and the output rate
5. VISUALIZATION AND USER INTERFACE

The user interface relies on the Comet and Bayeux protocols. Comet is a long-polling technique that is important to optimize latency effects in the client/server-side communication, avoiding closing connections during the message delivery unnecessarily.

Bayeux protocol implements a publish/subscribe paradigm on top of Comet, providing service channels that interface components might subscribe to. Every time a new event arrives, the web application publishes to a channel and the charts subscribed are updated in real-time.

A chart is fed with the output of a continuous query enabling event aggregation and processing. Different types of charts are available to users, such as those shown in Figures 7 and 8.

Figure 5: We illustrate the processing flow in the system: a query is used to join raw streams into a derived complex event that feeds our algorithm, which is then apt to detect anomalies in variables under surveillance.

Figure 6: Simplified sequence diagram of the update step.

Figure 7: Dashboard showing CPU processing capacity idle from 3 servers.

Figure 8: Number of slow queries in a database per server.
6. USE CASES

6.1 Big Brother Brasil

The system was used to monitor the Internet site of a Brazilian TV reality show, called Big Brother. The site is hosted at Globo.com, one of the largest ISP in Brazil. Every week the site receives thousands of users, voting to eliminate a participant of reality show. Indeed, the total numbers of votes in one elimination can exceed 70 millions with peaks of 13.5 thousands votes per minute.

The infrastructure used to support the voting application is similar to the presented in section 1.1: web servers, application servers, and databases. IT operators have also the challenging task of monitoring multiple information trying to solve, or anticipate, problems.

CEP engine was used to unify and aggregate information from business transactions, and IT metrics. This real-time information feeds a dashboard that was used by IT operation team. We had very good feedbacks about the ability to view aggregate information like the CPU consumption average of web servers together with business information, like the sum of processed votes in the last minute.

Another useful query used by IT operation to continuously monitor web server health was comparing the CPU consumption of one server with the average of all servers in a farm, that are expected to behave in a same range. We implemented this query creating a sliding time window containing CPU events of the last minute and triggering a alert event if a income CPU event is greater than 30% of average of events on window. The query is shown bellow in Figure 9:

```sql
SELECT avg(CPU.user), idle
FROM CPU.win:time[1 min] AS CPU
WHERE CPU.user > 1.3 avg(CPU.user)
```

Figure 9: EPL example to detect spike in CPU usage.

6.2 Network anomaly detection

Figure 10 shows the anomalies detected by FRAHST when analysing statistics from the data center routers, which are connected to five telecommunication operators by redundant network links. We monitored the number of bits per second in each link for both directions of communication averaged at each interval. In this incident, there was a communication failure with one of the operators, which caused two of the links to malfunction. The large failure was preceded by smaller loss of connectivity.

7. CONCLUSION

Correlation techniques has been historically applied to identify data center threats and root cause of incidents. Jakobson and Weissman [8] specified correlation operations that could be applied to correlate alarms in telecommunication. Fifteen years later [3], the IT infrastructure domain continues to be one of the major case studies on event correlation field. This fact enforces the relevance of the problem, and the huge opportunities that technology evolution brings to industry.

This paper shows a real case implementation that integrates CEP and machine learning in a Event-Driven ar-

chitecture to address the canonical IT monitoring problem. Our solution has been adopted for a real data center, where it has been shown to achieve high throughput and low latency. This use case has been presented at an international forum recently [2].

There are also future works for HOLMES planned based on the feedback of users. One challenge that needs to be addressed is the creation of interface to help IT operator to create and test queries to Esper [9] Another challenge is to integrate historical data with real-time data both for visualization and analysis.

8. REFERENCES


